R-Skript PUNO-Forschungsprojekt

Teil 1 – R, Datenmanagement, deskriptive Statistik

Dominik Vogel

Stand: 03.04.2019

Inhaltsverzeichnis

1 R installieren

Um mit `R` zu arbeiten, ist die Nutzung zweier Programme zu empfehlen: `R` (der Kern) und `RStudio` (als praktische Entwicklungsumgebung).

1. `R` herunterladen und installieren (in der Regel kann man die Standardeinstellungen „durchklicken“).
   - Für Windows: https://cran.r-project.org/bin/windows/base/
   - Für macOS: https://cran.r-project.org/bin/macosx/
2. `RStudio` herunterladen (https://www.rstudio.com/products/rstudio/download/#download) und installieren

1.1 R Studio konfigurieren

Es empfiehlt sich, einige Standardeinstellungen in `R` zu ändern. Klicken Sie hierfür auf `Tools` -> `Global Option ...` und nehmen Sie folgende Änderungen vor:

1. Unter `General` deaktivieren Sie die folgenden Optionen:
   1. Restore most recently opened project at startup
   2. Restore previously open source documents at startup
   3. Restore .RData into workspace at startup
   4. Stellen Sie „Save workspace to .RData on exit“ auf `Never`
2. Unter `Code` -> `Display` aktivieren Sie `Show margin` und stellen Sie den Wert auf 79
3. Unter `Code` -> `Saving` ändern Sie `Default text encoding` auf `UTF-8`

2 Hilfe finden

`R` ist nicht einfach und (fast) niemand kann sich alle benötigten Befehle und ihre Optionen merken. Umso wichtiger ist es, dass man weiß, wo man Hilfe findet.

2.1 Die R-Hilfe

Erster Anlaufpunkt ist in der Regel die `R`-Hilfe. Man erreicht sie direkt in `R`.

Weiß man bereits, wie ein Befehl heißt und will mehr dazu erfahren, bekommt man mit einem `?` gefolgt vom entsprechenden Befehl Zugriff auf die Dokumentation des Befehls:

```
?mean
```

Ohne genauen Befehl helfen zwei Fragezeichen und ein Suchbegriff in Anführungszeichen:

```
??"standard deviation"
```

2.2 Lehrbücher

Wenn die `R`-Hilfe nicht weiterhilft, kann vielleicht ein Buch helfen. Ich empfehle die folgenden Bücher:

- Field, A.; Miles, J.; Field, Z. (2012): Discovering Statistics Using R
- Hatzinger, R.; Hornik, K.; Nagel, H.; Maier, M. J. (2014): R – Einführung durch angewandte Statistik. (im Uni-Netz auch als ebook (http://lib.myilibrary.com?id=651009) verfügbar)
- Grolemund, G.; Wickham, H. (2016): R for Data Science. (auch als ebook (http://r4ds.had.co.nz/) frei verfügbar)
- Navarro, D.: Learning Statistics with R. (als ebook (https://learningstatisticswithr.com/) frei verfügbar)

2.3 Suchmaschine

In den meisten Fällen hilft auch die Suchmaschine des Vertrauens.

2.4 Stack Overflow

Hier leitet einen auch die Suchmaschine schnell hin: Stack Overflow (https://stackoverflow.com/questions/tagged/r). Auf der Plattform lässt sich für fast jedes `R`-Problem eine Lösung finden. Und wenn nicht, dann kann auch eine Frage gepostet werden.

3 Grundlagen

Die grundlegendste Verwendung von `R` ist der „Taschenrechner“. Geben Sie `5 + 7` in die „Console“ ein und drücken Sie Enter.

```
5 + 7
```

```
## [1] 12
```

Rechenoperatoren

Operator	Funktion
+	Addieren
-	Subtrahieren
*	Multiplizieren
/	Dividieren
^	Potenzieren

Ein Schritt weiter kommt man mit der Verwendung von Funktionen. Mit dem folgenden Befehl erhalten wir zum Beispiel den Exponenten von 1 (*Euler'sche Zahl*)

```
exp(1)
```

```
## [1] 2.718282
```

Funktionen lassen sich auch verschachteln.

```
log(exp(1))
```

```
## [1] 1
```

4 Arbeiten mit R

Ein Grundsatz von Forschung ist die Nachvollziehbarkeit des Forschungsprozesses. Ein anderer Forscher/eine andere Forscherin soll in der Lage sein, zu reproduzieren, was wir getan haben (Reproducability).

Dieser Grundsatz hilft aber auch dabei, in Teams zusammenzuarbeiten und zu einem späteren Zeitpunkt zu verstehen, was man getan hat.

> „Reproducibility is just collaboration with people you don't know – including yourself next week.“ — Philip Stark, University of California, Berkeley

Aus diesem Grund tippen wir den Code nicht in die Console, sondern speichern alles was wir machen in einem Skript. Und um dieses Skript beliebig kopieren zu können, ohne uns Gedanken darüber machen zu müssen, wo die Dateien liegen, betten wir das Skript in ein `RStudio`-Projekt ein. Das Projekt dient sozusagen als Container für alle Dateien des Projekts.

4.1 Projekte und Skripte

Zum Erstellen eines Projekts klicken Sie in `RStudio` auf `File` -> `New Project` und wählen im folgenden Dialogfenster `New Directory` -> `New Project` aus. Anschließend wählen Sie einen Namen für das Projekt (`Directory name`) sowie einen Speicherort und klicken Sie auf `Create Project`.

Erstellen Sie nun ein Skript in diesem Projekt: `File` -> `New File` -> `R Script`.

Beginnen Sie das neue Skript mit einer kurzen Beschreibung:

```
# Projekt: Übungsskript PUNO-Forschungsprojekt
# Autor: Dominik Vogel
# Datum: 04.04.2019
```

Alles was in `R` hinter einem `#` steht, ist ein Kommentar und wird beim Ausführen des Skripts ignoriert. Es dient nur dazu, dass Sie oder eine andere Person den Code versteht. Machen Sie reichlich Gebrauch davon!

Sie sollten Ihr Skript auch strukturieren. `RStudio` erkennt Kommentare die mit mindestens vier Strichen, Gleichzeichen oder Rauten enden als Überschriften. Am unteren Rand des Skriptfensters lässt sich damit navigieren.

```
# Überschrift ----------------------------
# Überschrift ============================
# Überschrift ############################
```

Hadley Wickham gibt auf seiner Seite (http://adv-r.had.co.nz/Style.html) viele weitere nützliche Hinweise zum Strukturieren und Formatieren von `R`-Code.

Speichern Sie nun Ihr neues Skript.

5 Objekte und Variablen

Da wir `R` nicht als Taschenrechner verwenden wollen, gehen wir weiter zu den Objekten.

> „To understand computations in R, two slogans are helpful: Everything that exists is an object. Everything that happens is a function call.“ (John M. Champers, einer der Entwickler von S, dem Vorgänger von R)

Kurzer Background: `R` ist eigentlich eine komplette „objektorientierte“ Programmiersprache. Das soll nicht erschrecken, hilft aber manchmal ein bisschen beim Verständnis. Es bedeutet, dass alles womit man in `R` hantiert ein Objekt ist. Dies kann ein einzelner Wert, ein ganzer Datensatz oder das Ergebnis einer Analyse sein. Die vorhandenen Objekte sieht man in `RStudio` rechts oben unter „Environment“.

Ein Objekt ist schnell erstellt, indem man einen Namen definiert und diesem mit `<-` einen Wert zuweist:

```
y <- 3
```

5.1 Bezeichnung von Objekten

`R` lässt relativ viel Freiraum bei der Benennung von Objekten. Es empfiehlt sich aber, einige Konventionen zu beachten, die die Arbeit erleichtern:

- Namen sollten Sinn ergeben und dennoch relativ kurz sein (da sie getippt werden müssen)
- Keine Leerzeichen verwenden
- Wörter können mit Punkt, Bindestrich oder Unterstrich getrennt werden. Ich bevorzuge den Unterstrich da dies auch in anderen Programmen Verwendung findet.
- Beschränken Sie sich auf Kleinbuchstaben, dann haben Sie beim Tippen weniger Mühe.

5.2 Variablen

Wir haben nun ein Objekt `y` erstellt, und ihm den Wert `3` zugewiesen. Ein solches Objekt mit einem einzelnen Wert wird als Variable, Skalar oder Value bezeichnet.

Man kann sich den Wert auch ausgeben lassen:

```
y
```

```
## [1] 3
```

Man kann natürlich auch mit Variablen rechnen.

```
y * 3
```

```
## [1] 9
```

```
x <- 5
x * y
```

```
## [1] 15
```

5.3 Vektoren

Man kann auch mehrere Werte in einem Objekt speichern. Man nennt dies einen Vektor.

Ein Vektor wird mit `c()` (*combine*) erstellt.

```
alter <- c(21, 78, 24, 26, 35)
alter
```

```
## [1] 21 78 24 26 35
```

Der Vektor `alter` hat nun fünf Elemente. Mit einem solchen Vektor kann man auch rechnen (vorausgesetzt er enthält nur numerische Elemente):

```
alter * 12
```

```
## [1] 252 936 288 312 420
```

`R` multipliziert in diesem Fall alle Werte des Vektors mit 12.

Auch erste Kennzahlen lassen sich mit einem Vektor ermitteln:

```
mean(alter)
```

```
## [1] 36.8
```

Wir können für einen Vektor auch Text verwenden, wenn wir diesen in Anführungszeichen setzen:

```
geschlecht <- c("maennlich", "weiblich")
geschlecht
```

```
## [1] "maennlich" "weiblich"
```

R kennt eine Vielzahl weiterer Objekttypen. Wir wollen an dieser Stelle nicht im Detail auf alle eingehen. Mehr zu den Objekttypen finden Sie im Zweifel in Teil 1.1. Da wir diese nur selten benötigen, gehen wir gleich zu den Datensätzen, da wir in aller Regel mit diesen arbeiten. Um dies möglichst einfach zu gestalten, lernen wir vorher, wie man Pakete installiert und Daten importiert.

6 Pakete

Eine der großen Stärken von R ist die Nutzung von s.g. Paketen. Diese sind Erweiterungen von R, die von jedem erstellt werden können. Die meisten davon, werden über ein zentrales Repository (CRAN) zur Verfügung gestellt und können direkt aus R heraus installiert werden:

```
install.packages("tidyverse", dep = TRUE)
```

Pakete müssen nur einmal installiert, nach einem Neustart von R jedoch neu geladen werden. Es empfiehlt sich daher alle verwendeten Pakete am Anfang eines Skripts zu laden:

```
library(tidyverse)
```

Pakete stellen in R neue Befehle (*Funktionen*) bereit. Im folgenden Abschnitt werden wir dies zum ersten Mal nutzen, wenn wir die Funktion `read_csv()` zum importieren von Daten verwenden. `read_csv()` gehört nicht zum Kern von R und lässt sich ohne das Paket `tidyverse` nicht nutzen.

7 Daten importieren

Da wir die Daten ja nicht selbst in R anlegen wollen, müssen wir diese in der Regel importieren. Dafür hält R viele verschiedene Funktionen bereit (mindestens eine pro Dateityp). In der Regel liegen die Daten als CSV-Datei vor. Um diese zu importieren nutzen wir die Funktion `read_csv()` aus dem `tidyverse` Paket.

7.1 Exkurs: Alternative Importfunktionen

Manchmal liegen die Daten auch in anderen Formaten vor. Hierfür eignen sich folgende Funktionen (im `tidyverse` Paket):

Funktion	Verwendung	Dateiendung
read_csv	Komma-getrennte Textdateien	.csv
read_csv2	Semikolon-getrennte Textdateien	.csv
read_excel	Excel-Dateien	.xls .xlsx
read_sav	SPSS-Dateien	.sav
read_dta	Stata-Dateien	.dta

7.2 Import

Wir starten mit dem „Lecturer-Datensatz“ von Field et al. (2012).

> „We took a random sample of five psychology lecturers from the University of Sussex [job = 1] and five psychology students [job = 0] and then measured how many friends they had, their weekly alcohol consumption (in units), their yearly income and how neurotic they were (higher score is more neurotic)“ (Field et al. 2012: 86).

Wir habe die Daten vorher unter `data/Lecturer_Data.csv` abgespeichert. Wir können diese anschließend mit `read_csv()` importieren.

```
lecturer_data <- read_csv("data/Lecturer_Data.csv")
```

```
## Parsed with column specification:
## cols(
##   ID = col_double(),
##   name = col_character(),
##   birth_date = col_character(),
##   job = col_double(),
##   friends = col_double(),
##   alcohol = col_double(),
##   income = col_double(),
##   neurotic = col_double()
## )
```

8 Datensätze

Wir haben nun ein neues Objekt mit dem Namen `lecturer_data` erstellt. Dieses ist ein Objekt vom Typ „Data Frame“[1] und enthält die importierten Daten. Wenn wir den Namen eingeben und ausführen, gibt `R` den Inhalt wieder.

```
lecturer_data
```

```
## # A tibble: 10 x 8
##       ID name   birth_date   job friends alcohol income neurotic
##    <dbl> <chr>  <chr>      <dbl>   <dbl>   <dbl>  <dbl>    <dbl>
##  1     1 Ben    7/3/1977       1       5      10  20000       10
##  2     2 Martin 5/24/1969      1       2      15  40000       17
##  3     3 Andy   6/21/1973      1       0      20  35000       14
##  4     4 Paul   7/16/1970      1       4       5  22000       13
##  5     5 Graham 10/10/1949     1       1      30  50000       21
##  6     6 Carina 11/5/1983      2      10      25   5000        7
##  7     7 Karina 10/8/1987      2      12      20    100       13
##  8     8 Doug   1/23/1989      2      15      16   3000        9
##  9     9 Mark   5/20/1973      2      12      17  10000       14
## 10    10 Zoe    11/12/1984     2      17      18     10       13
```

Die einzelnen Spalten eines Data Frames kann man mit einem `$` hinter dem Objektnamen ansprechen:

```
lecturer_data$alcohol
```

```
##  [1] 10 15 20  5 30 25 20 16 17 18
```

[1] Genau genommen handelt es sich um den Typ „Tibble“, der jedoch nur eine Spezialform eines Data Frames ist. Der Unterschied besteht im Wesentlichen darin, dass Tibble übersichtlicherer dargestellt werden.

```
lecturer_data$name
```

```
##  [1] "Ben"    "Martin" "Andy"   "Paul"   "Graham" "Carina" "Karina"
##  [8] "Doug"   "Mark"   "Zoe"
```

9 Daten inspizieren

Es gibt verschiedene Funktionen, um sich einen Eindruck von einem Data Frame zu machen.

- `head()` gibt die ersten 6 Zeilen aus
- `ncol()` gibt die Zahl der Variablen aus
- `nrow()` gibt die Zahl der Zeilen aus
- `colnames()` gibt die Namen der Spalten zurück
- `str()` gibt einen Überblick über die Struktur

```
str(lecturer_data)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 10 obs. of  8 variables:
##  $ ID        : num  1 2 3 4 5 6 7 8 9 10
##  $ name      : chr  "Ben" "Martin" "Andy" "Paul" ...
##  $ birth_date: chr  "7/3/1977" "5/24/1969" "6/21/1973" "7/16/1970" ...
##  $ job       : num  1 1 1 1 1 2 2 2 2 2
##  $ friends   : num  5 2 0 4 1 10 12 15 12 17
##  $ alcohol   : num  10 15 20 5 30 25 20 16 17 18
##  $ income    : num  20000 40000 35000 22000 50000 5000 100 3000 10000 10
##  $ neurotic  : num  10 17 14 13 21 7 13 9 14 13
##  - attr(*, "spec")=
##   .. cols(
##   ..   ID = col_double(),
##   ..   name = col_character(),
##   ..   birth_date = col_character(),
##   ..   job = col_double(),
##   ..   friends = col_double(),
##   ..   alcohol = col_double(),
##   ..   income = col_double(),
##   ..   neurotic = col_double()
##   .. )
```

Einen weiteren schönen Überblick liefert `skim()` aus dem Paket `skimr`. [*Leider ist der Output in der PDF nicht so schön wie in R, weil die Histogramme fehlen*]

```
install.packages("skimr", dep = TRUE)
```

```
library(skimr)
```

```
skim(lecturer_data)
```

```
## Skim summary statistics
##  n obs: 10
##  n variables: 8
##
## -- Variable type:character -----------------------------------------------------------------
##    variable missing complete  n min max empty n_unique
##  birth_date       0       10 10   8  10     0       10
```

```
##        name       0       10 10   3   6     0       10
##
## -- Variable type:numeric -------------------------------------------------------------------------
##  variable missing complete  n    mean       sd p0     p25     p50      p75
##   alcohol       0       10 10    17.6     7.04  5   15.25    17.5    20
##   friends       0       10 10     7.8     6.14  0    2.5      7.5    12
##        ID       0       10 10     5.5     3.03  1    3.25     5.5     7.75
##    income       0       10 10 18511   18001.35 10 3500    15000   31750
##       job       0       10 10     1.5     0.53  1    1        1.5     2
##  neurotic       0       10 10    13.1     3.98  7   10.75    13      14
##   p100
##     30
##     17
##     10
##  50000
##      2
##     21
```

9.1 Missings

Jedes Statistiktool geht anders mit fehlenden Werten (*Missings*) um. `R` stellt Missings als `NA` (not available) dar. Eine besondere Form davon sind unmögliche Werte `NaN` (not a number). Dies gibt `R` beispielsweise beim Dividieren durch Null aus. Mehr zu Missings können Sie unter http://www.statmethods.net/input/missingdata.html nachlesen.

Laden wir den Lecturer Datensatz noch einmal in einer Variante mit Missings.

```
lecturer_data_mi <- read_csv("data/Lecturer_Data_Missing.csv")
```

```
## Parsed with column specification:
## cols(
##   ID = col_double(),
##   name = col_character(),
##   birth_date = col_character(),
##   job = col_double(),
##   friends = col_double(),
##   alcohol = col_double(),
##   income = col_double(),
##   neurotic = col_double()
## )
```

Beim Inspizieren sehen wir die Missings.

```
lecturer_data_mi
```

```
## # A tibble: 10 x 8
##       ID name   birth_date   job friends alcohol income neurotic
##    <dbl> <chr>  <chr>      <dbl>   <dbl>   <dbl>  <dbl>    <dbl>
##  1     1 Ben    7/3/1977       1       5      10  20000       10
##  2     2 Martin 5/24/1969      1       2      15  40000       17
##  3     3 Andy   6/21/1973      1       0      20  35000       14
##  4     4 Paul   7/16/1970      1       4       5  22000       13
##  5     5 Graham 10/10/1949     1       1      30  50000       21
##  6     6 Carina 11/5/1983      2      10      25   5000       NA
##  7     7 Karina 10/8/1987      2      12      20    100       13
##  8     8 Doug   1/23/1989      2      15      16   3000        9
##  9     9 Mark   5/20/1973      2      12      17     NA       14
## 10    10 Zoe    11/12/1984     2      17      18     10       13
```

Im Gegensatz zu vielen anderen Statistiktools ignoriert R die Missings nicht automatisch. Wir merken das zum Beispiel, wenn wir einen Mittelwert berechnen wollen.

```
mean(lecturer_data$income)
```

```
## [1] 18511
```

```
mean(lecturer_data_mi$income)
```

```
## [1] NA
```

Die Lösung besteht darin, R anzuweisen, die Missings zu ignorieren.

```
mean(lecturer_data_mi$income, na.rm = TRUE)
```

```
## [1] 19456.67
```

10 Daten auswählen

In R führen viele Wege nach Rom. Beim Auswählen und Manipulieren von Daten sind es besonders viele. Wir nutzen hier den Weg, den das Paket `dplyr`, das auch zum `tidyverse` gehört, zur Verfügung stellt.

Um `dplyr` dazu zu bewegen, uns mitzuteilen, welche Änderungen vorgenommen wurden, laden wir noch das kleine Zusatzpaket `tidylog`

```
install.packages("tidylog", dep = TRUE)
library(tidylog)
```

Wir nutzen zum Auswählen von Daten bevorzugt die Funktionen `filter()` und `select()` .

10.1 Fälle auswählen mit filter()

Mit `filter()` kann man einzelne Zeilen (= Fälle) auswählen und auch direkt mehrere Bedingungen (die mit einem logischen UND geknüpft werden) angeben.

```
filter(lecturer_data, job == 1, alcohol > 10)
```

```
## filter: removed 7 out of 10 rows (70%)
```

```
## # A tibble: 3 x 8
##      ID name   birth_date   job friends alcohol income neurotic
##   <dbl> <chr>  <chr>      <dbl>   <dbl>   <dbl>  <dbl>    <dbl>
## 1     2 Martin 5/24/1969      1       2      15  40000       17
## 2     3 Andy   6/21/1973      1       0      20  35000       14
## 3     5 Graham 10/10/1949     1       1      30  50000       21
```

Um das Ergebnis in einem neuen Objekt zu speichern, müssen wir dies explizit angeben:

```
drinking_lecturers <- filter(lecturer_data, job == 1, alcohol > 10)
```

```
## filter: removed 7 out of 10 rows (70%)
```

```
drinking_lecturers
```

```
## # A tibble: 3 x 8
##      ID name   birth_date   job friends alcohol income neurotic
##   <dbl> <chr>  <chr>      <dbl>   <dbl>   <dbl>  <dbl>    <dbl>
## 1     2 Martin 5/24/1969      1       2      15  40000       17
## 2     3 Andy   6/21/1973      1       0      20  35000       14
## 3     5 Graham 10/10/1949     1       1      30  50000       21
```

`filter()` lässt sich auch auf character Variablen anwenden.

```
filter(lecturer_data, name == "Martin")
```

```
## filter: removed 9 out of 10 rows (90%)
```

```
## # A tibble: 1 x 8
##      ID name   birth_date   job friends alcohol income neurotic
##   <dbl> <chr>  <chr>      <dbl>   <dbl>   <dbl>  <dbl>    <dbl>
## 1     2 Martin 5/24/1969      1       2      15  40000       17
```

10.2 Fälle mit Missings ausschließen – drop_na()

`tidyverse` bietet uns eine komfortable Funktion, um Zeilen (= Fälle) mit fehlenden Werten auszuschließen. Die Funktion heißt `drop_na()`. Man kann entweder jene Fälle ausschließen, die auf einer oder mehreren vorgegebenen Variablen fehlende Werte besitzt oder alle Fälle, die auf einer beliebigen Variable fehlende Werte aufweisen.

```
# Fälle mit fehlenden Werten auf job und income ausschließen
# (Zeile 9 wird gelöscht)
drop_na(lecturer_data_mi, job, income)
```

```
## # A tibble: 9 x 8
##      ID name   birth_date   job friends alcohol income neurotic
##   <dbl> <chr>  <chr>      <dbl>   <dbl>   <dbl>  <dbl>    <dbl>
## 1     1 Ben    7/3/1977       1       5      10  20000       10
## 2     2 Martin 5/24/1969      1       2      15  40000       17
## 3     3 Andy   6/21/1973      1       0      20  35000       14
## 4     4 Paul   7/16/1970      1       4       5  22000       13
## 5     5 Graham 10/10/1949     1       1      30  50000       21
## 6     6 Carina 11/5/1983      2      10      25   5000       NA
## 7     7 Karina 10/8/1987      2      12      20    100       13
## 8     8 Doug   1/23/1989      2      15      16   3000        9
## 9    10 Zoe    11/12/1984     2      17      18     10       13
```

```
# Alle Fälle ausschließen, die mindestens einen fehlenden Wert aufweisen
# (Zeilen 6 und 9 werden gelöscht)
drop_na(lecturer_data_mi)
```

```
## # A tibble: 8 x 8
##      ID name   birth_date   job friends alcohol income neurotic
##   <dbl> <chr>  <chr>      <dbl>   <dbl>   <dbl>  <dbl>    <dbl>
## 1     1 Ben    7/3/1977       1       5      10  20000       10
## 2     2 Martin 5/24/1969      1       2      15  40000       17
## 3     3 Andy   6/21/1973      1       0      20  35000       14
```

```
## 4     4 Paul   7/16/1970      1       4       5  22000       13
## 5     5 Graham 10/10/1949     1       1      30  50000       21
## 6     7 Karina 10/8/1987      2      12      20    100       13
## 7     8 Doug   1/23/1989      2      15      16   3000        9
## 8    10 Zoe    11/12/1984     2      17      18     10       13
```

10.3 Variablen auswählen mit select()

Einzelne Spalten (= Variablen) lassen sich mit `select()` auswählen.

```
select(lecturer_data, name, job, alcohol)
```

```
## select: dropped 5 variables (ID, birth_date, friends, income, neurotic)
```

```
## # A tibble: 10 x 3
##    name     job alcohol
##    <chr>  <dbl>   <dbl>
##  1 Ben        1      10
##  2 Martin     1      15
##  3 Andy       1      20
##  4 Paul       1       5
##  5 Graham     1      30
##  6 Carina     2      25
##  7 Karina     2      20
##  8 Doug       2      16
##  9 Mark       2      17
## 10 Zoe        2      18
```

`select()` bietet außerdem die Möglichkeit Zusatzfunktionen einzusetzen, um Variablen auszuwählen. Bspw.:

- `starts_with("bcd")`: wählt alle Variablen aus, die mit „bcd“ beginnen.
- `ends_with("xyz")`: wählt alle Vairablen aus, die mit „xyz“ enden.
- `contains("hjk")`: wählt alle Variablen aus, die „hjk“ im Namen haben.
- `num_range("x", 1:3)`: wählt die Variablen x1, x2 und x3 aus.

Mit `select()` kann man auch bestimmte Variablen ausschließen:

```
select(lecturer_data, -neurotic)
```

```
## select: dropped one variable (neurotic)
```

```
## # A tibble: 10 x 7
##       ID name   birth_date   job friends alcohol income
##    <dbl> <chr>  <chr>      <dbl>   <dbl>   <dbl>  <dbl>
##  1     1 Ben    7/3/1977       1       5      10  20000
##  2     2 Martin 5/24/1969      1       2      15  40000
##  3     3 Andy   6/21/1973      1       0      20  35000
##  4     4 Paul   7/16/1970      1       4       5  22000
##  5     5 Graham 10/10/1949     1       1      30  50000
##  6     6 Carina 11/5/1983      2      10      25   5000
##  7     7 Karina 10/8/1987      2      12      20    100
##  8     8 Doug   1/23/1989      2      15      16   3000
##  9     9 Mark   5/20/1973      2      12      17  10000
## 10    10 Zoe    11/12/1984     2      17      18     10
```

10.4 Einschub: Vergleichsoperatoren

Weiter oben haben wir bereits die Rechenoperatoren (`+`, `-`, `*`, usw.) kennengelernt. Für die Auswahl von Fällen und Variablen benötigen wir noch die Vergleichsoperatoren:

Operator	Bedeutung
<	kleiner als
<=	kleiner gleich
>	größer als
>=	größer gleich
==	ist gleich
!=	ist nicht gleich
|	oder
&	und

11 Variablen manipulieren

11.1 Variablen umbenennen

Oft wollen wir Variablen auch umbenennen. Wir testen dies Anhand eines neuen Datensatzes. `fevs_2014_subsample` ist eine Zufallsstichprobe aus dem Federal Employee Viewpoint Survey von 2014.

```
fevs <- read_csv("data/fevs_2014_subsample.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   agency = col_character(),
##   plevel1 = col_character(),
##   plevel2 = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
head(fevs, n = 2) # Nur die ersten beiden Zeilen ausgeben
```

```
## # A tibble: 2 x 101
##      id  userid postwt agency plevel1 plevel2    q1    q2    q3    q4    q5
##   <dbl>   <dbl>  <dbl> <chr>  <chr>   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl>
## 1   118 1.06e11   14.9 NV     NV60    <NA>        4     4    NA     4     4
## 2   232 1.59e11   21.1 AR     ARBA    <NA>        2     2     1     2     4
## # ... with 90 more variables: q6 <dbl>, q7 <dbl>, q8 <dbl>, q9 <dbl>,
## #   q10 <dbl>, q11 <dbl>, q12 <dbl>, q13 <dbl>, q14 <dbl>, q15 <dbl>,
## #   q16 <dbl>, q17 <dbl>, q18 <dbl>, q19 <dbl>, q20 <dbl>, q21 <dbl>,
## #   q22 <dbl>, q23 <dbl>, q24 <dbl>, q25 <dbl>, q26 <dbl>, q27 <dbl>,
## #   q28 <dbl>, q29 <dbl>, q30 <dbl>, q31 <dbl>, q32 <dbl>, q33 <dbl>,
## #   q34 <dbl>, q35 <dbl>, q36 <dbl>, q37 <dbl>, q38 <dbl>, q39 <dbl>,
## #   q40 <dbl>, q41 <dbl>, q42 <dbl>, q43 <dbl>, q44 <dbl>, q45 <dbl>,
## #   q46 <dbl>, q47 <dbl>, q48 <dbl>, q49 <dbl>, q50 <dbl>, q51 <dbl>,
## #   q52 <dbl>, q53 <dbl>, q54 <dbl>, q55 <dbl>, q56 <dbl>, q57 <dbl>,
## #   q58 <dbl>, q59 <dbl>, q60 <dbl>, q61 <dbl>, q62 <dbl>, q63 <dbl>,
## #   q64 <dbl>, q65 <dbl>, q66 <dbl>, q67 <dbl>, q68 <dbl>, q69 <dbl>,
## #   q70 <dbl>, q71 <dbl>, q72 <dbl>, q73 <dbl>, q74 <dbl>, q75 <dbl>,
## #   q76 <dbl>, q77 <dbl>, q78 <dbl>, q79 <dbl>, q80 <dbl>, q81 <dbl>,
## #   q82 <dbl>, q83 <dbl>, q84 <dbl>, minority <dbl>, super <dbl>,
## #   sex <dbl>, agegrp <dbl>, paycat <dbl>, fedten <dbl>, leaving <dbl>,
## #   retire <dbl>, dis <dbl>, mil <dbl>, edu <dbl>
```

Wir widmen uns den drei Variablen `q40`, `q69` und `q71`.

> q40: I recommend my organization as a good place to work.
>
> q69: Considering everything, how satisfied are you with your job?
>
> q71: Considering everything, how satisfied are you with your organization?
>
> (1 = very dissatisfied; 5 = very satisfied)

Zum Umbenennen nutzen wir `rename()` aus `dplyr`:

```
fevs <- rename(fevs, recommend_orga = q40)
fevs <- rename(fevs, sat_job = q69)
fevs <- rename(fevs, sat_orga = q71)
```

Für `rename()` gibt man also immer zuerst den Datensatz an und dann den neuen und alten Namen mit einem `=` getrennt: `rename(Datensatz, neu = alt)`

11.2 Variablen generieren

In den wenigsten Fällen begnügen wir uns mit den Variablen, die ein Datensatz enthält. Wir wollen neue generieren.

Wir wollen nun die Variable `job_satisfaction` generieren. Diese soll die Summe der Antworten aus den Variablen `q40`, `q69` und `q71` sein.

Zum Erstellen einer neue Variable nutzen wir die Funktion `mutate()` aus `dplyr`. `mutate()` fügt am Ende („rechts“) des Datensatzes eine neue Variable ein. Das Ergebnis muss mit `<-` in einem Objekt gespeichert werden, da `R` ansonsten lediglich das Ergebnis (Datensatz mit neuer Variable) in der Console ausgibt.

```
fevs <- mutate(fevs, job_satisfaction = recommend_orga + sat_job + sat_orga)
```

```
## mutate: new variable 'job_satisfaction' with 14 unique values and 6% NA
```

```
fevs$job_satisfaction[1:10] # Werte 1 bis 10 ausgeben
```

```
##  [1] 10  6 12 12 12  5 11  8 14 15
```

Wir können auch einen Mittelwertindex bilden. Hierfür nutzen wir die Funktion `rowMeans()`:

```
M1 <- select(fevs, recommend_orga, sat_job, sat_orga) # Variablen auswählen
```

```
## select: dropped 99 variables (id, userid, postwt, agency, plevel1, ...)
```

```
job_satisfaction_mean <- rowMeans(M1, na.rm = FALSE) # Mittelwert berechnen
fevs <- data.frame(fevs, job_satisfaction_mean) # Mittelwert in fevs einfügen
fevs$job_satisfaction_mean[1:10] # Werte 1 bis 10 ausgeben
```

```
##  [1] 3.333333 2.000000 4.000000 4.000000 4.000000 1.666667 3.666667
##  [8] 2.666667 4.666667 5.000000
```

Wir können auch mit Bedingungen arbeiten. Hierfür gibt es die Funktion `ifelse()`. Die Grundstruktur ist `ifelse(Wenn, Dann, Sonst)`. Beispiel:

```
x <- 1:6
x
```

```
## [1] 1 2 3 4 5 6
```

```
ifelse(x < 4, "A", "B")
```

```
## [1] "A" "A" "A" "B" "B" "B"
```

Die Funktion prüft also, ob der Wert von x kleiner 4 ist und gibt A aus, wenn dies zutrifft und B, wenn nicht. In Kurzform: *Wenn $x < 4$, dann A, sonst B.*

Die Funktion können wir in Verbindung mit `mutate()` auch zum Generieren von Variablen nutzen. Erstellen wir zur Illustration eine Variable, die den Wert 1 annimmt, wenn `job_satisfaction_mean` größer gleich 4 ist und den Wert 0 annimmt, wenn nicht.

```
fevs <- mutate(fevs, high_job_sati = ifelse(job_satisfaction_mean >= 4, 1, 0))
```

```
## mutate: new variable 'high_job_sati' with 3 unique values and 6% NA
```

```
fevs$high_job_sati[1:10]
```

```
##  [1] 0 0 1 1 1 0 0 0 1 1
```

11.3 Einschub: Reliabilität mit Cronbachs Alpha

Eben haben wir mehrere Items zu einer Variable zusammengefasst. Ob dies im vorliegenden Fall ein angemessenes Vorgehen ist, muss allerdings getestet werden. Die Items sollen schließlich „zueinander passen“. Das bedeutet, die Items sollen dasselbe Konstrukt messen. Hierfür ist es erforderlich, dass die Messung reliabel ist. Wir wollen, dass jemand mit einer geringen Arbeitszufriedenheit auf unserer Variable einen niedrigen Wert hat und jemand mit einer hohen Arbeitszufriedenheit einen hohen Wert. Hierzu müssen die verwendeten Items möglichst dasselbe messen, nämlich die Arbeitszufriedenheit. Um dies zu testen, setzen wir Cronbachs Alpha ein. Dieser gibt im Wesentlichen an, wie stark eine bestimmte Menge von Items covariieren. Die Details hierzu finden sich in Section 17.8 von Field et al. (2012).

Testen wir also, ob die vorher verwendeten Items geeignet sind, um ein gemeinsames Konstrukt zu messen. Hierfür verwenden wir die Funktion `alpha()` aus dem Paket `psych`. Bevor wir die Funktion verwenden, müssen wir allerdings noch ein neues Objekt erstellen, dass nur die gewünschten Items enthält.

```
install.packages("psych", dep = TRUE)
#library(psych)
```

```
M1 <- select(fevs, recommend_orga, sat_job, sat_orga) # Variablen auswählen
```

```
## select: dropped 101 variables (id, userid, postwt, agency, plevel1, ...)
```

```
psych::alpha(M1)
```

```
##
## Reliability analysis
## Call: psych::alpha(x = M1)
##
##   raw_alpha std.alpha G6(smc) average_r S/N    ase mean sd median_r
##        0.9       0.9    0.86      0.75 8.9 0.0051  3.5  1     0.77
##
##  lower alpha upper     95% confidence boundaries
## 0.89 0.9 0.91
##
##  Reliability if an item is dropped:
##               raw_alpha std.alpha G6(smc) average_r S/N alpha se var.r
```

```
## recommend_orga      0.87      0.87    0.77      0.77 6.6   0.0076    NA
## sat_job             0.87      0.87    0.77      0.77 6.6   0.0076    NA
## sat_orga            0.83      0.83    0.71      0.71 4.8   0.0100    NA
##                 med.r
## recommend_orga  0.77
## sat_job         0.77
## sat_orga        0.71
##
##  Item statistics
##                    n raw.r std.r r.cor r.drop mean  sd
## recommend_orga 1150  0.91  0.90  0.83   0.78  3.6 1.1
## sat_job        1118  0.90  0.90  0.83   0.78  3.6 1.1
## sat_orga       1119  0.93  0.93  0.88   0.83  3.4 1.1
##
## Non missing response frequency for each item
##                    1    2    3    4    5 miss
## recommend_orga 0.05 0.13 0.21 0.41 0.21 0.02
## sat_job        0.05 0.13 0.18 0.44 0.20 0.05
## sat_orga       0.06 0.17 0.22 0.40 0.14 0.05
```

Im Befehl setzten wir das `psych` mit zwei Doppelpunkten vor die Funktion `alpha` da es möglicherweisen zu Konflikten mit der Funktion `alpha` aus `ggplot2` (ein Teil von `tidyverse`) kommen kann.

Wir sind nun vor allem an dem Wert `raw_alpha` interessiert. Dieser beträgt 0,9. Allgemein wird als Daumenregel akzeptiert, dass α mindestens 0,7 betragen sollte. Wir können hier mit einem sehr guten Wert von 0,9 also problemlos ein gemeinsames Konstrukt bilden.

Kleiner Hinweis: `alpha` prüft standardmäßig nicht, ob sich in der Auswahl einzelne Items befinden, die umgekehrt (reversed) codiert sind. Dies kann mit der Option `check.keys = TRUE` aktiviert werden. Ein kleines Beispiel:

```
fevs$sat_job2 <- 6 - fevs$sat_job # sat_job "umdrehen"
M1 <- select(fevs, recommend_orga, sat_job2, sat_orga) # Variablen auswählen
```

```
## select: dropped 102 variables (id, userid, postwt, agency, plevel1, ...)
```

```
psych::alpha(M1, check.keys = TRUE)
```

```
## Warning in psych::alpha(M1, check.keys = TRUE): Some items were negatively correlated with total
##  This is indicated by a negative sign for the variable name.
```

```
##
## Reliability analysis
## Call: psych::alpha(x = M1, check.keys = TRUE)
##
##   raw_alpha std.alpha G6(smc) average_r S/N    ase mean sd median_r
##        0.9       0.9    0.86      0.75 8.9 0.0051  3.5  1     0.77
##
##  lower alpha upper     95% confidence boundaries
## 0.89 0.9 0.91
##
##  Reliability if an item is dropped:
##                raw_alpha std.alpha G6(smc) average_r S/N alpha se var.r
## recommend_orga      0.87      0.87    0.77      0.77 6.6   0.0076    NA
## sat_job2-           0.87      0.87    0.77      0.77 6.6   0.0076    NA
## sat_orga            0.83      0.83    0.71      0.71 4.8   0.0100    NA
##                 med.r
## recommend_orga  0.77
```

```
## sat_job2-       0.77
## sat_orga        0.71
##
##  Item statistics
##                    n raw.r std.r r.cor r.drop mean  sd
## recommend_orga 1150  0.91  0.90  0.83   0.78  3.6 1.1
## sat_job2-      1118  0.90  0.90  0.83   0.78  3.6 1.1
## sat_orga       1119  0.93  0.93  0.88   0.83  3.4 1.1
##
## Non missing response frequency for each item
##                    1    2    3    4    5 miss
## recommend_orga 0.05 0.13 0.21 0.41 0.21 0.02
## sat_job2       0.20 0.44 0.18 0.13 0.05 0.05
## sat_orga       0.06 0.17 0.22 0.40 0.14 0.05
```

`alpha` gibt hier direkt am Anfang eine Warnung aus und markiert das automatisch umgedrehte Item mit einem - hinter dem Namen.

12 Deskriptive Statistik

Für das folgende Kapitel nutzen wir Daten zur Prüfungsangst von Studierenden, die von Field et al. (2012) stammen.

Code: a number indicating from which participant the scores came.

Revise: the total hours spent revising.

Exam: mark on the exam as a percentage.

Anxiety: the score on the Exam Anxiety Questionnaire.

Gender: whether the participant was male or female (stored as strings of text).

```
exam <- read_csv("data/exam_anxiety.csv")
```

```
## Parsed with column specification:
## cols(
##   code = col_double(),
##   revise = col_double(),
##   exam = col_double(),
##   anxiety = col_double(),
##   gender = col_character()
## )
```

```
exam
```

```
## # A tibble: 103 x 5
##     code revise  exam anxiety gender
##    <dbl>  <dbl> <dbl>   <dbl> <chr>
##  1     1      4    40    86.3 Male
##  2     2     11    65    88.7 Female
##  3     3     27    80    70.2 Male
##  4     4     53    80    61.3 Male
##  5     5      4    40    89.5 Male
##  6     6     22    70    60.5 Female
##  7     7     16    20    81.5 Female
##  8     8     21    55    75.8 Female
##  9     9     25    50    69.4 Female
## 10    10     18    40    82.3 Female
```

```
## # ... with 93 more rows
```

12.1 Mittelwert und ähnliches

Wir interessieren uns zunächst für den Anxiety Score und wollen ein paar Kennwerte hierzu betrachten.

```
mean(exam$anxiety)
```

```
## [1] 74.34367
```

```
median(exam$anxiety)
```

```
## [1] 79.044
```

```
min(exam$anxiety)
```

```
## [1] 0.056
```

```
max(exam$anxiety)
```

```
## [1] 97.582
```

```
sd(exam$anxiety)
```

```
## [1] 17.18186
```

```
IQR(exam$anxiety)
```

```
## [1] 14.911
```

Mithilfe der `quantile`-Funktion lassen sich Quantile bestimmen:

```
quantile(exam$anxiety, 0.5) # 50%-Quantil = Median
```

```
##    50%
## 79.044
```

```
quantile(exam$anxiety, c(0.25, 0.75)) # 25% und 75% Quantil
```

```
##    25%    75%
## 69.775 84.686
```

```r
quantile(exam$anxiety, seq(0.1, 1, by = 0.1)) #10% bis 100% Quantil
```

```
##     10%     20%     30%     40%     50%     60%     70%     80%     90%
## 57.6044 66.3092 71.7900 75.0140 79.0440 81.6232 83.3964 85.6532 89.5220
##    100%
## 97.5820
```

Einen schnellen Überblick über eine Reihe von Kennwerten liefert `summary()`:

```r
summary(exam$anxiety)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.056  69.775  79.044  74.344  84.686  97.582
```

Noch mehr Werte liefert `describe()` aus dem Paket `psych`:

```r
psych::describe(exam$anxiety)
```

```
##    vars   n  mean    sd median trimmed   mad  min   max range  skew
## X1    1 103 74.34 17.18  79.04   77.05 10.75 0.06 97.58 97.53 -1.95
##    kurtosis   se
## X1     4.73 1.69
```

`describe()` kann auch auf einen ganzen Data Frame angewandt werden:

```r
psych::describe(exam)
```

```
## Warning in psych::describe(exam): NAs durch Umwandlung erzeugt
```

```
## Warning in FUN(newX[, i], ...): kein nicht-fehlendes Argument für min; gebe
## Inf zurück
```

```
## Warning in FUN(newX[, i], ...): kein nicht-fehlendes Argument für max; gebe
## -Inf zurück
```

```
##         vars   n  mean    sd median trimmed   mad  min    max  range  skew
## code       1 103 52.00 29.88  52.00   52.00 38.55 1.00 103.00 102.00  0.00
## revise     2 103 19.85 18.16  15.00   16.70 11.86 0.00  98.00  98.00  1.95
## exam       3 103 56.57 25.94  60.00   57.75 29.65 2.00 100.00  98.00 -0.36
## anxiety    4 103 74.34 17.18  79.04   77.05 10.75 0.06  97.58  97.53 -1.95
## gender*    5 103   NaN    NA     NA     NaN    NA  Inf   -Inf   -Inf    NA
##         kurtosis   se
## code       -1.24 2.94
## revise      4.34 1.79
## exam       -0.91 2.56
## anxiety     4.73 1.69
## gender*       NA   NA
```

Das bereits erwähnte `skim()` aus `skimr` ist natürlich auch hilfreich

```r
skim(exam)
```

```
## Skim summary statistics
##  n obs: 103
```

```
##  n variables: 5
##
## -- Variable type:character ---------------------------------------------------------------------
##  variable missing complete   n min max empty n_unique
##    gender       0      103 103   4   6     0        2
##
## -- Variable type:numeric -----------------------------------------------------------------------
##  variable missing complete   n  mean    sd    p0   p25   p50   p75   p100
##   anxiety       0      103 103 74.34 17.18 0.056 69.78 79.04 84.69  97.58
##      code       0      103 103 52    29.88 1     26.5  52    77.5  103
##      exam       0      103 103 56.57 25.94 2     40    60    80    100
##    revise       0      103 103 19.85 18.16 0      8    15    23.5   98
```

12.2 Häufigkeitsverteilungen

Bei nominalen oder ordinalen Variablen sind Häufigkeitstabellen aufschlussreich.

```
table(exam$gender)
```

```
##
## Female   Male
##     51     52
```

Prozentuale Anteile erhält man mit `prop.table()`. Man muss jedoch eine Tabelle übergeben. Wir erledigen das in einem Zug:

```
prop.table(table(exam$gender))
```

```
##
##    Female      Male
## 0.4951456 0.5048544
```

Mit `table` lassen sich auch Kreuztabellen erstellen. Hierzu verwenden wir einen Dummy, der angibt, ob der Anxiety Score über 90 liegt.

```
exam <- mutate(exam, anxiety_high = ifelse(anxiety > 90, 1, 0))
```

```
## mutate: new variable 'anxiety_high' with 2 unique values and 0% NA
```

```
table(exam$anxiety_high)
```

```
##
##  0  1
## 93 10
```

Für die Kreuztabelle geben wir nun nur noch die beiden Variablen an

```
table(exam$anxiety_high, exam$gender)
```

```
##
##     Female Male
##   0     46   47
##   1      5    5
```

Noch interessanter sind hingegen oftmals die Verteilungen von metrischen Variablen auf bestimmte Gruppen. In unserem Fall zum Beispiel die Prüfungsangst getrennt nach Geschlecht. Hierfür bietet uns `R` die Funktion `tapply()`. `tapply()` erwartet drei Argumente: Eine metrische Variable, eine kategoriale Variable und eine Funktion die angewandt werden soll.

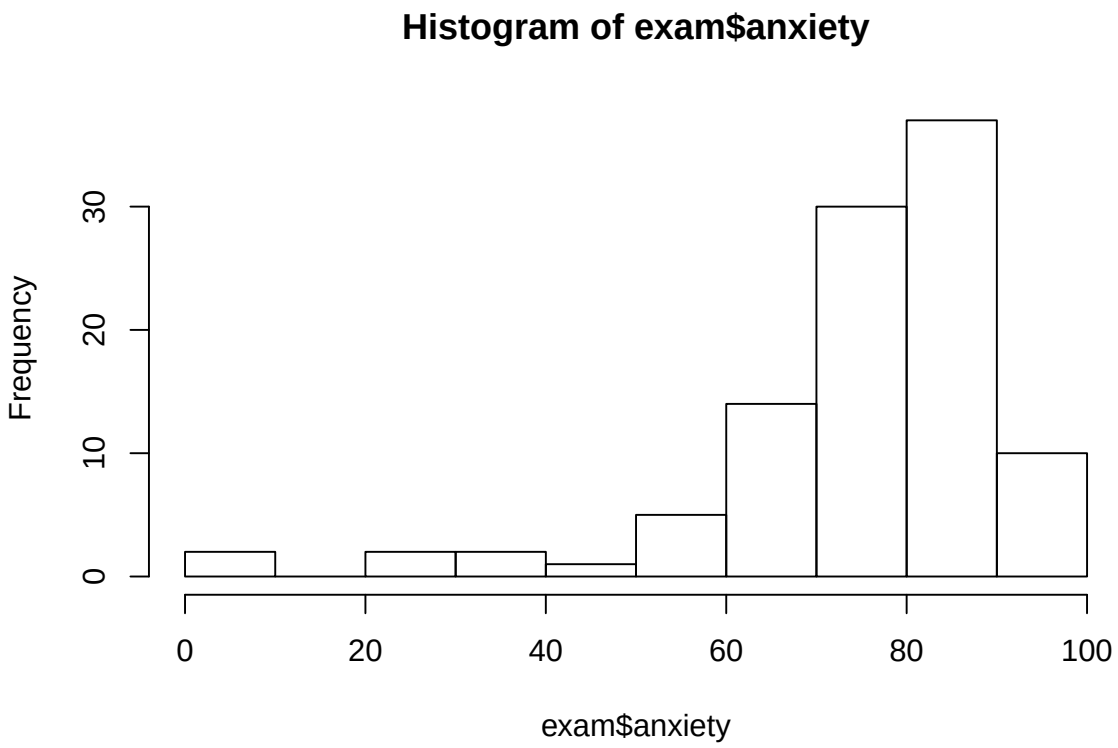
```
tapply(exam$anxiety, exam$gender, mean)
```

```
##   Female     Male
## 74.30282 74.38373
```

Der Unterschied scheint also auf den ersten Blick gering zu sein.

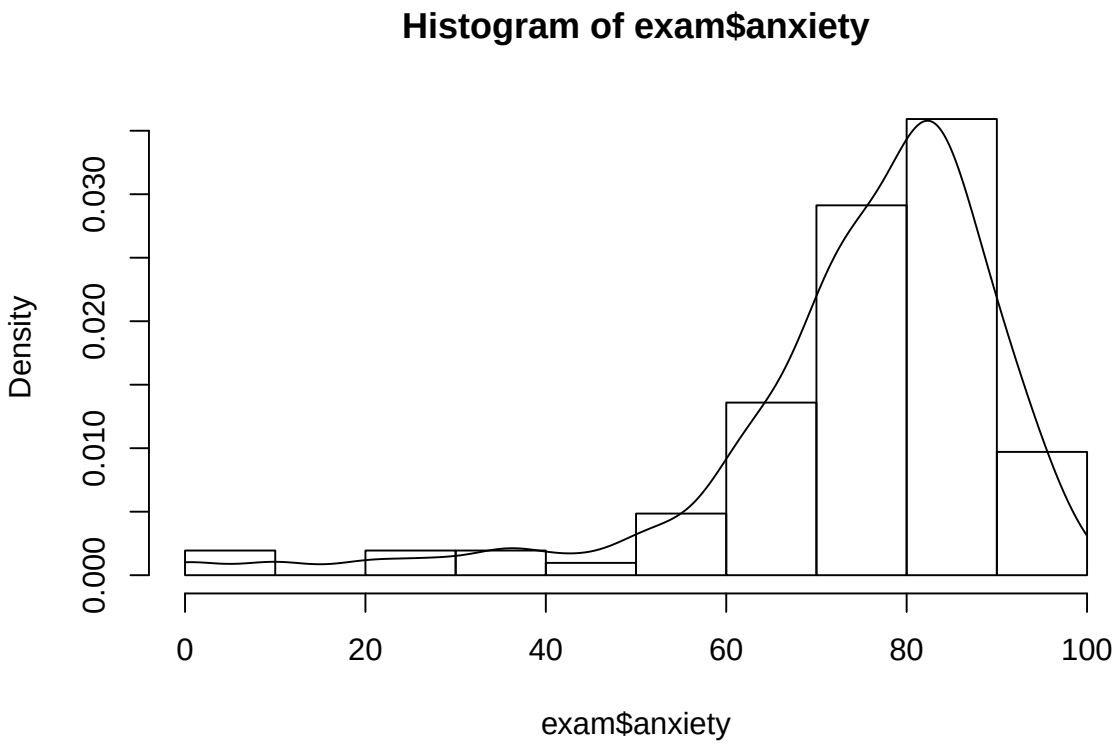
Wir können einzelne Variablen natürlich auch graphisch betrachten:

```
hist(exam$anxiety)
```



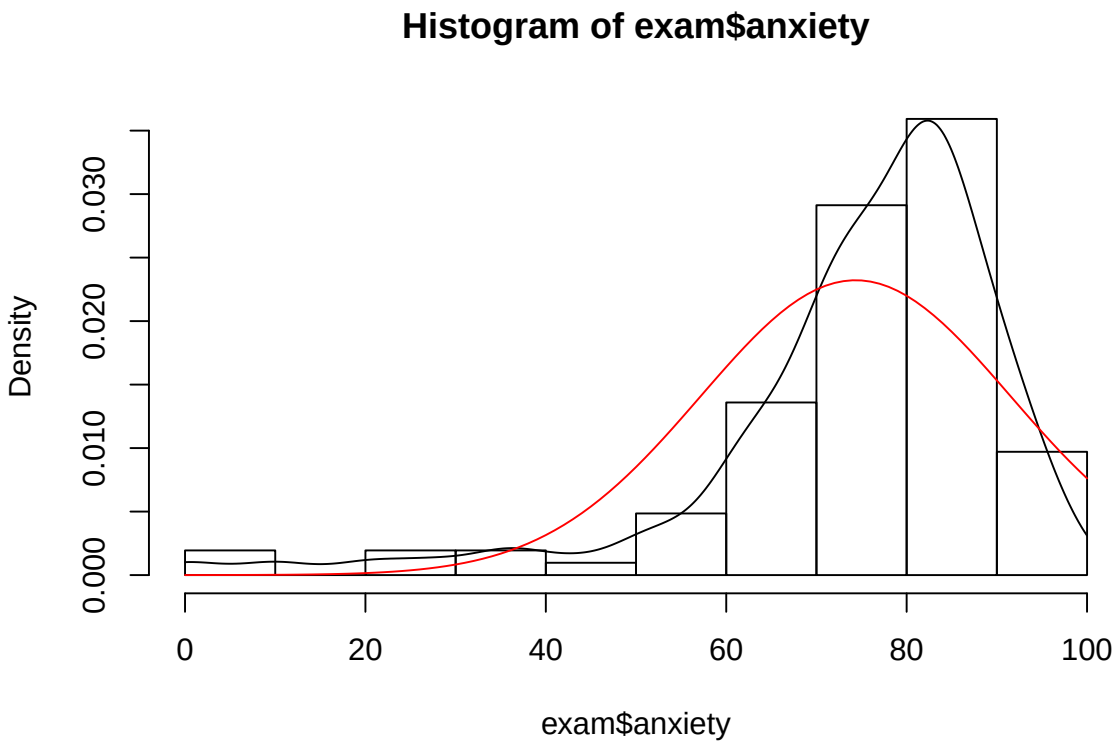
Wir können auch eine Kern-Dichte-Funktion hinzufügen, um abzuschätzen, wie die Verteilung als Kurve aussehen würde.

```
hist(exam$anxiety, freq = FALSE)
lines(density(exam$anxiety, from = 0, to = 100))
```



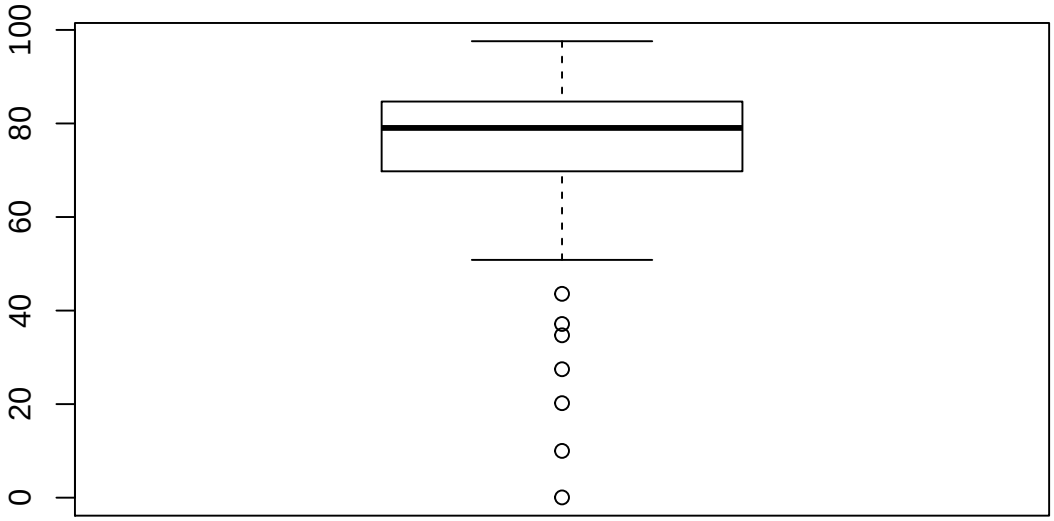
Auch der Vergleich mit einer Normalverteilung ist möglich:

```
hist(exam$anxiety, freq = FALSE)
lines(density(exam$anxiety, from = 0, to = 100))
curve(dnorm(x,
            mean = mean(exam$anxiety),
            sd = sd(exam$anxiety)),
      add = TRUE,
      col = "red")
```



Auch ein Boxplot ist möglich

```
boxplot(exam$anxiety)
```



Viele weitere Möglichkeiten zur Visualisierung finden Sie in Teil 4.